PSis - Programação de Sistemas 2013/2014
Exame 2ª Época, 28 de Junho de 2014, 11h30, Duração: 2h

1. [1.0 valor] Descreva de que modo a multi-programação permite acelerar a execução de duas aplicações concorrentes num computador com um único processador.
Em que caso é que não se obtêm ganhos da execução concorrente de duas aplicações?

Quando uma aplicação está bloqueada (por exemplo à espera de dados da rede, teclado ou disco) outra aplicação pode estar a ser executada.
A multiprogramação não oferece ganhos quando as aplicações a concorrer pelo processador não se bloqueiam (por exemplo não fazem I/O), nestes casos a mudança de contexto introduz penalizações).

2. [1.0 valores] Quando se implementa um servidor concorrente, a gestão de *threads* pode ser realizada de diversos modos: *pool* de *threads* vs criação dinâmica.
Indique uma vantagem e um inconveniente de se implementar um *pool* de *threads* em comparação com a criação dinâmica.
Na sua implementação mais simples uma *pool* tem um tamanho limitado. Descreva de que modo se pode contornar esta característica de forma eficiente.

O uso da pool de threads obriga a um programação mais complexa (ao nível da gestão das threads e sincronização).
O uso de pool de threads acelera o tratamento dos pedidos, visto não se perder tempo a criar a nova thtred.
O sistema pode criar dinamicamente threads quando a pool está vazia (todas a threads estão a processar pedidos). Quando estas novas threads terminam é necessário mantê-las na pool ou destruí-las.

3. [1.0 valor] Descreva genericamente os mecanismos de comunicação por memória partilhada e por núcleo. Indique duas diferenças qualitativas (vantagem de uma/desvantagem da outra) entre estas duas aproximações.

Por memória partilhada os programas escrevem/lêem os dados em variáveis partilhadas.
Por núcleos os programa invocam chamadas de sistema que permitem enviar e receber os dados (pipes, sockets).

Memória partilhada	***Núcleo***
Mais rápida	*Mais lenta*
Difícil de programar	*Fácil de programar*
Necessita de sincronizaçcão	*Não necessita de sincronizaçcão*

PSis - Programação de Sistemas 2013/2014
Exame 2ª Época, 28 de Junho de 2014, 11h30, Duração: 2h

4. [1.0 valor] Usando *pipe(s)*, implemente as primitivas de **iniciar**, **esperar** e **assinalar** de um *mutex*. O *mutex* é criado desbloqueado. O(s) *pipe(s)* usados podem ser variáveis globais.

```
int pipefd[2];


void iniciar(){
      int n=1;
      pipe(pipefd);
      write(pipefd[1], &n, sizeof(n));
}
void esperar(){
      int n;
      read(pipefd[0], &n, sizeof(n));
}
void assinalar(){
      int n=1;
      write(pipefd[1], &n, sizeof(n));
}
```

5. [1.0 valor] A instrução atómica *testAndSet* permite implementar regiões críticas com o seguinte código:

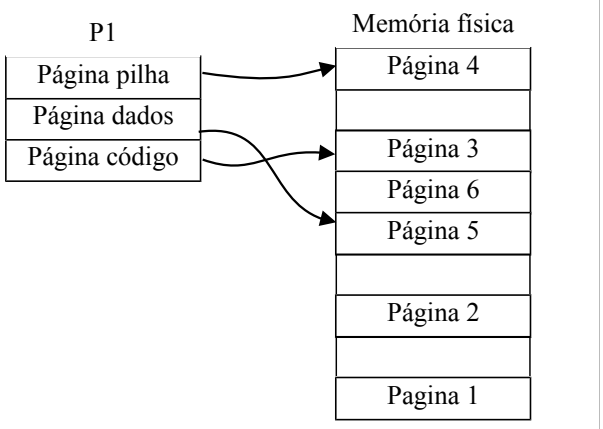
```
while ( TestAndSet(&lock) );
 /* Região crítica */
lock = FALSE;
```

Descreva dois problemas desta solução.

Espera activa, o processo está a "gastar" processador quando verifica se pode entrar na região crítica.
Não é garantido que um processo que está à espera de entrar, entre. O algoritmo de escalonamento pode levar a que tal processo nunca seja escalonado quando a variável tem o valor FALSE.

PSis - Programação de Sistemas 2013/2014
Exame 2ª Época, 28 de Junho de 2014, 11h30, Duração: 2h

6. [1.0 valor] A seguinte figura mostra a utilização de memória de um processo antes do *fork*.




```
…

i=10;
fork();
i++;
printf("%d", i);

…
```

Com a ajuda da figura, descreva como é feita a gestão e alocação de memória quando é realizado um *fork*. Represente graficamente (de modo semelhante à figura anterior) a configuração de memória dos dois processos (P1 - pai, P2 - filho) imediatamente após o *fork*.

Assuma que o `i++` do processo filho (P2) executa antes do `i++` do processo pai (P1). Descreva o que ocorre quando é realizada essa instrução. Represente graficamente a configuração de memória dos dois processos imediatamente após esse `i++` no processo filho (P2).

7. [1.0 valor] Em UNIX (um sistema com memória virtual) o acesso a uma posição de memória (leitura ou escrita) pode gerar uma falha de página (*page fault*).
Indique os dois casos em que tal ocorre e, para cada caso, descreva o que ocorre e de que modo a falha é resolvida.

O endereço de memória é inválido e não corresponde a uma página existente.
É gerado o sinal segmentation fault (que pode ser tratado pela aplicação).
O endereço corresponde a uma página não existente em memória, mas sim no disco.
O sistema operativo verifica que há espaço em memória para essa página.
Se não houver espaço, uma das páginas em memória é copiada para o disco.
A página é copiada de disco para memória.
Programa continua.

(continua na página seguinte)

PSis - Programação de Sistemas 2013/2014
Exame 2ª Época, 28 de Junho de 2014, 11h30, Duração: 2h

Resolva esta questão num folha de exame diferente. Preencha essa folha com o seu nome e número.

7. [3 valores] Considere um sistema de reserva de lugares num cruzeiro. Existem 3 tipos de lugares: L1, L2 e L3 sendo o número máximo de lugares MAX (incluindo todas as categorias). O número máximo de lugares de tipo L3 é MAX3, em que MAX3 é menor que MAX. Os lugares dos restantes tipos não têm limite específico.

O sistema de reservas em causa suporta dois tipos de pedidos: **reserva** e **desiste** de um lugar de um determinado tipo.

- Quando é feito um pedido por um processo (ou uma thread) de um lugar de tipo L1 ou L2, o processo só deverá ficar bloqueado se os MAX lugares já tiverem sido todos reservados.

- Quando é feito um pedido por um processo (ou uma thread) de um lugar de tipo L3, o processo só deverá ficar bloqueado se os MAX lugares ou os MAX3 lugares já tiverem sido todos reservados.

- Qualquer destes processos (ou threads) só podem ser desbloqueados quando se reunirem as condições para proceder à reserva do lugar respectivo.

Note que pode suceder que os números de lugares reservados do tipo L1 e L2 seja tal que o número restante de lugares que sobram seja inferior a MAX3.

Utilizando semáforos com as habituais operações de **esperar** e **assinalar**, programe em pseudo-código as funções:

- `reserva(int tipo)` e
- `desiste(int tipo)`.

Solução:

```
sem_t lugares = MAX
sem_t lugares3 = NMAX3

void reserva(int tipo){
     if (tipo==3)
          esperar(lugares3);
     esperar(lugares);
}

void desiste(int tipo){
     assinalar(lugares);
     if (tipo==3)
          assinalar(lugares3);
}
```